A Hardware Algorithm for Computing Reciprocal Square Root

Naofumi Takagi
Department of Information Engineering
Nagoya University
Nagoya 464-8603, Japan
email: ntakagi@nuie.nagoya-u.ac.jp

Abstract

A hardware algorithm for computing the reciprocal square root which appears frequently in multimedia and graphics applications is proposed. The reciprocal square root is computed by iteration of carry-propagation-free additions, shifts, and multiplications by one digit. Different specific versions of the algorithm are possible, depending on the radix, the redundancy factor of the digit set, and etc. Each version of the algorithm can be implemented as a sequential (folded) circuit or a combinational (unfolded) circuit, which has a regular array structure suitable for VLSI.

1 Introduction

Reciprocal square root (or inverse square root) appears frequently in multimedia and graphics applications. Several microprocessors, such as MIPS R10000/R12000 [1, 2] and IBM PowerPC 740/750 [3], as well as graphic engines, have reciprocal square root instruction in their instruction set. This instruction is realized by a division (a reciprocal computation) followed by a square rooting (or vice versa) using a divide/square-root unit based on digit-recurrence methods, or by a convergence method such as Newton-Raphson method using a multiplier. The former realization is very slow, because its latency is the sum of those of division and square rooting. On the other hand, by the latter realization, the instruction occupies the multiplier for more than ten clock cycles and may decrease the performance of a system.

The increased significance of multimedia and graphics applications and the recent advances of VLSI technologies make it attractive to accelerate reciprocal square root operation by special hardware. Several methods using table look-up and multiplication were proposed for this operation, in the last decade [4, 5, 6, 7].

In this paper, we propose a digit-recurrence algorithm for computing reciprocal square root, which does not require multiplication. By providing a circuit based on the algorithm, we can release the multiplier from the computation and, hence, increase the performance of a system, greatly. It is also nice to avoid intermediate rounding errors between atomic operations, but only have a bounded error for the final result.

For reciprocal square rooting, as square rooting [8], we can derive digit-recurrence equations of the residual and of the partial result. However, the recurrence equation of the residual includes the product of the partial result and the operand and, therefore, a naive calculation of the j-th residual requires a j-digit by n-digit multiplication. In the algorithm to be proposed, we keep the product of the partial result and the operand, as well as the partial result itself, and derive recurrence equations of residual and of the product, so that we can calculate them by addition/subtractions, shifts, and multiplications by one digit. We select each reciprocal square root digit from a redundant digit set by estimating the residual and the product. We perform addition/subtractions appearing in the calculation of the recurrence equations of the residual and of the product without carry/borrow propagation by representing them in a redundant representation such as the carry-save form.

We can design different specific versions of the algorithm, depending on the radix, the redundancy factor of the reciprocal square root digit set, the type of representation of the residual and the product, the digit selection function, and etc., as digit-recurrence algorithms for division or square rooting [8]. We can implement each version as a sequential (folded) circuit or a combinational (unfolded) circuit, which has a regular array structure suitable for VLSI.

In the rest of this paper, we will first propose a general algorithm and make implementation consideration in Section 2. We will show a radix-2 version of the algorithm and its sequential implementation in Section 3, and will show a radix-4 version and its sequential implementation in Section 4.

0-7695-1150-3/01 $10.00 © 2001 IEEE

2 General algorithm

We consider computation of the reciprocal square root of the mantissa part of a floating-point number. We compute $S = X^{-\frac{1}{2}}$, where $\frac{1}{4} < X < 1$. We assume X is represented as an n-digit r-ary fraction where $r = 2^b$. We intend to obtain S so that it satisfies

$$-r^{-n} < X^{-\frac{1}{2}} - S < r^{-n}. \quad (1)$$

Note that $1 < X^{-\frac{1}{2}} < 2$. We assume that S is represented as an $n+1$-digit r-ary number.

2.1 Algorithm

As digit-recurrence algorithms for division or square rooting [8], the reciprocal square root digit q_j is obtained step by step. Let $S[j]$ be the partial result after j iterations. Then, $S[j] = S[0] + \sum_{i=1}^{j} q_i r^{-i}$, where $S[0]$ is the initial value of the partial result. The recurrence equation of the partial result is

$$S[j+1] = S[j] + q_{j+1} r^{-j-1}. \quad (2)$$

We select q_{j+1} from a redundant digit set $\{-a, \cdots, -1, 0, 1, \cdots, a\}$, where $\frac{r}{2} \le a < r$. The final result is $S = S[n] = S[0] + \sum_{i=1}^{n} q_i r^{-i}$. The result has to be computed for n-digit precision. Namely, (1) must hold.

We define a residual (or scaled partial remainder) $W[j]$ as

$$W[j] = r^j(1 - X \cdot S[j]^2). \quad (3)$$

Substituting $j+1$ for j in (3), we get $W[j+1] = r^{j+1}(1 - X \cdot S[j+1]^2)$. Then, from (2) and (3), we get the recurrence equation of the residual as

$$W[j+1] = rW[j] - 2X \cdot S[j] q_{j+1} - X q_{j+1}^2 r^{-j-1}. \quad (4)$$

Since this equation includes the term $-2X \cdot S[j]q_{j+1}$, we need multiplication of an n-digit number X and a j-digit number $S[j]$ for the calculation. To avoid the multiplication, we keep $X \cdot S[j]$.

We define $P[j]$ as $X \cdot S[j]$. Then, the recurrence equation of $W[j]$, (4), is rewritten as

$$W[j+1] = rW[j] - 2P[j] q_{j+1} - X q_{j+1}^2 r^{-j-1}. \quad (5)$$

The recurrence equation of $P[j]$ is

$$P[j+1] = P[j] + X q_{j+1} r^{-j-1}. \quad (6)$$

In order to obtain S that satisfies (1), we have to bound $W[j]$ within a certain range. (1) is rewritten as

$$(S - r^{-n})^2 < X^{-1} < (S + r^{-n})^2.$$

Since $S = S[j] + \sum_{i=j+1}^{n} q_i r^{-i}$ and the minimum and the maximum reciprocal square root digit values are $-a$ and a, respectively, it is further rewritten as

$$(S[j] - \rho r^{-j})^2 < X^{-1} < (S[j] + \rho r^{-j})^2,$$

where $\rho = \frac{a}{r-1}$ is the redundancy factor of the reciprocal square root digit set. Therefore, from (3), we get the bounds for $W[j]$ as

$$-2X \cdot S[j]\rho + X\rho^2 r^{-j} < W[j] < 2X \cdot S[j]\rho + X\rho^2 r^{-j}. \quad (7)$$

At the beginning of the computation, (7) has to be satisfied for $j = 0$. Hence,

$$\begin{aligned} -2X \cdot S[0]\rho + X\rho^2 &< W[0] = 1 - X \cdot S[0]^2 \\ &< 2X \cdot S[0]\rho + X\rho^2 \end{aligned} \quad (8)$$

have to be satisfied. Since $\frac{1}{2} < \rho \le 1$, we can satisfy (8) by, for example, letting $S[0] = \frac{3}{2}$ and $W[0] = 1 - \frac{9}{4}X$. When $\rho = 1$, we may also let $S[0] = 1$ and $W[0] = 1 - X$.

The algorithm for computing the reciprocal square root consists in performing n iterations of calculation of the recurrence equations (2), (5), and (6). The general algorithm is summarized as follows.

Algorithm [RSQRT]
Step 1:
Set the appropriate values to $S[0]$, $W[0]$ and $P[0]$;
Step 2:
for $j := 0$ to $n-1$ do
{
Select q_{j+1} from $\{-a, \cdots, -1, 0, 1, \cdots, a\}$;
$S[j+1] := S[j] + q_{j+1} r^{-j-1}$;
$W[j+1] := rW[j] - (2P[j] + Xq_{j+1} r^{-j-1}) q_{j+1}$;
$P[j+1] := P[j] + Xq_{j+1} r^{-j-1}$;
}

$X^{-\frac{1}{2}}$ is obtained as $S[n]$. Since $S[n]$ is in the r-ary signed-digit representation, we have to convert it into ordinary (non-redundant) r-ary representation. We can apply the on-the-fly conversion method [9] to this conversion. When we adopt the on-the-fly conversion, we keep the non-redundant representation of $S[j]$, $S[j]^=$, and that of $S[j] - r^{-j}$, $S[j]^-$, in the computation. Note that $X^{\frac{1}{2}}$ is also obtained as $P[n]$.

We can increase the speed of the implementation with a small increase in hardware complexity by performing the addition/subtractions in the recurrence equations for the residual and for the product without carry/borrow propagation by the use of a redundant representation. Therefore, in this paper, we concentrate on this type of implementations. Namely, we represent the residual $W[j]$, as well as the product $P[j]$, in a redundant representation, such as

the carry-save form, and perform the addition/subtractions without carry/borrow propagation.

Now, we consider the selection of reciprocal square root digit, q_{j+1}. We have to select q_{j+1} from $\{-a, \cdots, -1, 0, 1, \cdots, a\}$ so that $W[j+1]$ satisfies

$$-2X \cdot S[j+1]\rho + X\rho^2 r^{-j-1} < W[j+1] < 2X \cdot S[j+1]\rho + X\rho^2 r^{-j-1}.$$

q_{j+1} depends on the shifted residual $rW[j]$, the operand X and the partial result $S[j]$.

Let the interval of $rW[j]$ where k $(= -a, -a+1, \cdots, a)$ can be selected as q_{j+1} be $L_k[j] < rW[j] < U_k[j]$. Then,

$$L_k[j] = 2X \cdot S[j](k-\rho) + X(k-\rho)^2 r^{-j-1}, \quad (9)$$

$$U_k[j] = 2X \cdot S[j](k+\rho) + X(k+\rho)^2 r^{-j-1}. \quad (10)$$

Note that the lower bound of the interval for $k = -a$ and the upper bound of the interval for $k = a$ are equal to the lower bound and the upper bound of $rW[j]$, respectively.

The continuity condition of adjacent intervals, i.e., $U_{k-1}[j] > L_k[j]$, yields

$$(2\rho - 1)(2X \cdot S[j] + X(2k-1)r^{-j-1}) > 0, \quad (11)$$

which is always satisfied. The left-hand side of (11) gives the overlap between adjacent selection intervals. Using the overlap, we can select q_{j+1} by estimates of $rW[j]$, X, and $S[j]$.

Since $X \cdot S[j] = P[j]$, (9) and (10) are rewritten as

$$L_k[j] = 2P[j](k-\rho) + X(k-\rho)^2 r^{-j-1}, \quad (12)$$

$$U_k[j] = 2P[j](k+\rho) + X(k+\rho)^2 r^{-j-1}, \quad (13)$$

respectively. Then, we can select q_{j+1} by estimates of $rW[j]$, X, and $P[j]$. Since the second terms of the right-hand sides of (12) and (13) decrease very rapidly as j increases, we can make the digit selection function independent of X except for a few, if any, small j's.

Let the estimates of $rW[j]$ and $P[j]$ be $r\hat{W}[j]$ and $\hat{P}[j]$, respectively. We obtain $r\hat{W}[j]$ by truncating $rW[j]$ to t fractional bits, and obtain $\hat{P}[j]$ by truncating $P[j]$ to d fractional bits. (Note that not r-ary digits but bits.) The digit selection function is described by a set of threshold values $\{m_k(\hat{P}[j]) | k = -a+1, -a+2, \cdots, a\}$. k is selected as q_{j+1} if $m_k(\hat{P}[j]) \le r\hat{W}[j] < m_{k+1}(\hat{P}[j])$.

When $W[j]$ is in the carry-save form,

$$r\hat{W}[j] \le rW[j] < r\hat{W}[j] + 2^{-t+1}.$$

Therefore,

$$m_k(\hat{P}[j]) > \max_{\hat{P}[j]}(L_k[j])$$

and

$$(m_k(\hat{P}[j]) - 2^{-t}) + 2^{-t+1} \le \min_{\hat{P}[j]}(U_{k-1}[j]),$$

i.e.,

$$\max_{\hat{P}[j]}(L_k[j]) < m_k(\hat{P}[j]) \le \min_{\hat{P}[j]}(U_{k-1}[j]) - 2^{-t} \quad (14)$$

have to be satisfied. $\max_{\hat{P}[j]}(L_k[j])$ denotes the maximum value of the lower bound of the interval of $rW[j]$ where k can be selected as q_{j+1} when the estimate of $P[j]$ is $\hat{P}[j]$. $m_k(\hat{P}[j])$ must be a multiple of 2^{-t} that satisfies (14). Note that the maximum value of $r\hat{W}[j]$ for which $k-1$ is selected as q_{j+1} is $m_k(\hat{P}[j]) - 2^{-t}$.

The minimum overlap required for a feasible digit selection is

$$\min_{\hat{P}[j]}(U_{k-1}[j]) - \max_{\hat{P}[j]}(L_k[j]) > 2^{-t}. \quad (15)$$

When $P[j]$ is in the carry-save form, $\hat{P}[j] \le P[j] < \hat{P}[j] + 2^{-d+1}$. Therefore, from (12) and (13), for $k > 0$,

$$\begin{aligned} \max_{\hat{P}[j]}(L_k[j]) &< 2(\hat{P}[j] + 2^{-d+1})(k-\rho) + X(k-\rho)^2 r^{-j-1}, \\ \min_{\hat{P}[j]}(U_{k-1}[j]) &= 2\hat{P}[j](k-1+\rho) + X(k-1+\rho)^2 r^{-j-1}. \end{aligned} \quad (16)$$

For $k \le 0$

$$\begin{aligned} \max_{\hat{P}[j]}(L_k[j]) &= 2\hat{P}[j](k-\rho) + X(k-\rho)^2 r^{-j-1}, \\ \min_{\hat{P}[j]}(U_{k-1}[j]) &> 2(\hat{P}[j] + 2^{-d+1})(k-1+\rho) \\ &\quad + X(k-1+\rho)^2 r^{-j-1}. \end{aligned} \quad (17)$$

These expressions are used to determine the digit selection function, i.e., the threshold values. Since they depend on j, a different selection function might result for different j. In practical cases, we can get a common selection function except for a few, if any, small j's. This will be illustrated later, in the cases $r = 2$ and $r = 4$.

We can design different specific versions of the algorithm, depending on the radix r, the redundancy factor ρ of the reciprocal square root digit set, the type of representation of the residual and the product (carry-save or signed-digit), the digit selection function, and etc.

2.2 Implementation consideration

We can implement each version of the algorithm as a sequential (folded) circuit or a combinational (unfolded) circuit. We can also use pipelining.

First, we consider a circuit for performing one iteration of Step 2. It consists of the following modules as shown in Fig. 1. We assume that residual $W[j]$ and the product $P[j]$ are represented in the carry-save form and that the on-the-fly conversion is adopted.

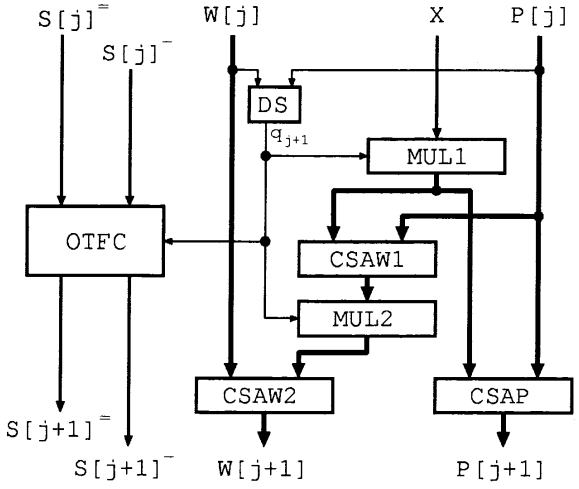

Figure 1. Block diagram of a circuit for computing one iteration of Step 2

1. DS: a reciprocal square root digit selector for selecting q_{j+1} by examining significant bits of $rW[j]$ and $P[j]$.
2. MUL1: a multiplier for multiplying (shifted) X by q_{j+1} and producing $Xq_{j+1}r^{-j-1}$.
3. CSAW1: carry-save adders for adding $2P[j]$ and the product of MUL1.
4. MUL2: a multiplier for multiplying the output of CSAW1 by q_{j+1}.
5. CSAW2: carry-save adders for adding $rW[j]$ and the product of MUL2 and producing $W[j+1]$.
6. CSAP: carry-save adders for adding $P[j]$ and the product of MUL1 and producing $P[j+1]$.
7. OTFC: an on-the-fly converter for calculating $S[j+1]^=$ and $S[j+1]^-$ which mainly consists of selectors.

When we implement a reciprocal square rooting circuit as a sequential circuit which performs one iteration of Step 2 in each clock cycle, it consists of a combinational circuit part and registers. The combinational circuit part is the circuit shown in Fig. 1, with a simple additional circuit for setting initial values to the registers. We need registers REG-SE and REG-SM, REG-WC and REG-WS, REG-PC and REG-PS, and REG-X, for storing $S[j]^=$ and $S[j]^-$, $W[j]$, $P[j]$, and X, respectively. Since $W[j]$ and $P[j]$ are in the carry-save form, we need two registers for storing them each. In order to avoid variable ($j+1$-digit) shift of X, we keep Xr^{-j} in REG-X by 1-digit shifting in each clock cycle.

The circuit performs the computation in $n+1$ clock cycles. The cycle time is a constant independent of n. The amount of hardware is proportional to n. It has a regular structure with digit-slice feature suitable for VLSI implementation.

Of course, we can construct a sequential circuit which performs more than one iteration of Step 2 per clock cycle.

When we implement a reciprocal square rooting circuit as a combinational circuit, it is constructed by connecting a simple circuit for performing Step 1 and n copies of the circuit for one iteration of Step 2 discussed above, in series. Shifts are implemented by wiring. The delay (logical depth) of the circuit is proportional to n. The amount of hardware is proportional to n^2. It has a regular 2-dimensional cellular array structure suitable for VLSI implementation.

3 A radix-2 version

We can design different specific versions of the algorithm. In this section, we show the details of a radix-2 version of the algorithm and consider its sequential implementation.

3.1 Algorithm

Here, we consider the case that the radix r is 2, the reciprocal square root digit set is $\{-1, 0, 1\}$ (i.e., $a = 1, \rho = 1$) and the residual $W[j]$ and the product $P[j]$ is represented in the carry-save form.

The recurrence equations are

$$\begin{aligned} S[j+1] &= S[j] + q_{j+1}2^{-j-1}, \\ W[j+1] &= 2W[j] - (2P[j] + Xq_{j+1}2^{-j-1})q_{j+1}, \\ P[j+1] &= P[j] + Xq_{j+1}2^{-j-1}. \end{aligned}$$

Since $\rho = 1$, at the beginning of the computation, we can let $S[0] = 1$, $W[0] = 1 - X$, and $P[0] = X$ for all X. Then, $0 < W[0] < \frac{3}{4}$.

To obtain a digit selection function, we get the values of $L_k[j]$ and $U_k[j]$ from (9) and (10).

$$\begin{aligned} U_{-1}[j] &= 0 \\ L_0[j] &= -2X \cdot S[j] + 2^{-j-1}X \\ U_0[j] &= 2X \cdot S[j] + 2^{-j-1}X \\ L_1[j] &= 0 \end{aligned}$$

Since $X > \frac{1}{4}$, $S[j] \geq 1$, and $j \geq 0$,

$$L_0[j] < -\frac{3}{8}$$
$$U_0[j] > \frac{1}{2}.$$

These values are independent of $P[j]$, X, and j. From (14), $-\frac{3}{8} \leq m_0 \leq -2^{-t}$ and $0 < m_1 \leq \frac{1}{2} - 2^{-t}$ have to be satisfied.

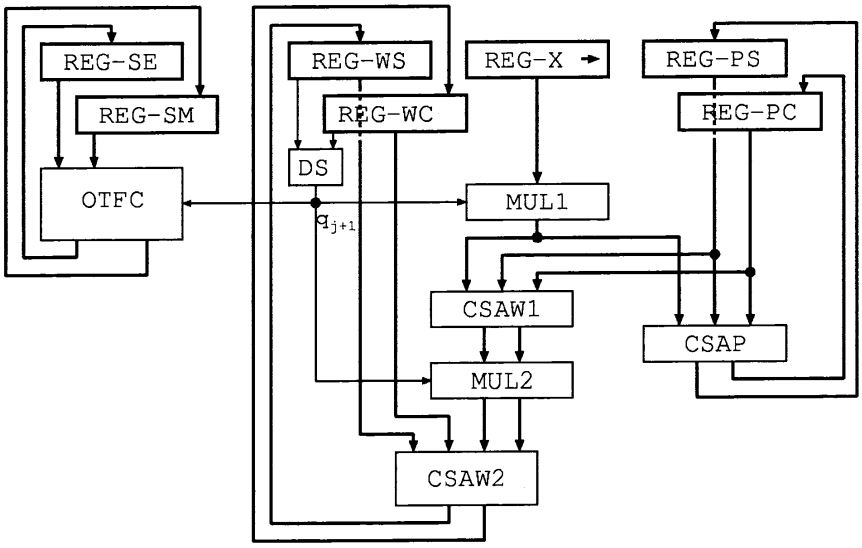

Figure 2. A block diagram of a sequential implementation of the radix-2 version

Therefore, We get $m_0 = -\frac{1}{4}$ and $m_1 = \frac{1}{4}$ by letting $t = 2$, which are independent of j.

This radix-2 version of the algorithm is summarized as follows. We assume that the on-the-fly conversion is adopted. Since $-2 < W[j] < 3$, $W[j]$ is represented in a two's complement carry-save form with 3-bit integer part (including the sign bit). Since $0 < P[j] < 2$, $P[j]$ is represented in an unsigned carry-save form with 1-bit integer part. Therefore, we can select q_{j+1} by examining the most significant 6 (carry-save) bits of $2W[j]$.

Algorithm [RSQRT_R2]

Step 1:

$S[0]^= := 1$; $S[0]^- := 0$; $W[0] := 1 - X$; $P[0] := X$;

Step 2:

for $j := 0$ to $n - 1$ do

{

$$q_{j+1} := \begin{cases} -1 & if \quad 2\hat{W}[j] \le -\frac{1}{2} \\ 0 & if \quad -\frac{1}{4} \le 2\hat{W}[j] \le 0 \\ 1 & if \quad \frac{1}{4} \le 2\hat{W}[j] \end{cases};$$

($2\hat{W}[j]$: truncation of $2W[j]$ to 2 fractional bits.)

Calculate $S[j+1]^=$ and $S[j+1]^-$;

(On-the-fly conversion.)

$W[j+1] := 2W[j] - (2P[j] + Xq_{j+1}2^{-j-1})q_{j+1}$;

(Carry-save additions.)

$P[j+1] := P[j] + Xq_{j+1}2^{-j-1}$;

(Carry-save addition.)

}

$S[n]^=$ is the result.

3.2 Sequential implementation

Here, we consider implementation of the radix-2 version as a sequential circuit which performs one iteration of Step 2 in each clock cycle. It consists of several combinational circuit modules and registers as shown in Fig. 2.

The digit selector, DS, is a combination of a 6-bit carry-propagate adder and simple constant value comparators. A buffer for driving q_{j+1} is also required. The on-the-fly converter, OTFC, mainly consists of a pair of 2-to-1 selectors. MUL1 and MUL2 are multiplexers, which produce 0 or the data input itself or its complement according to the value of q_{j+1}. Each of CSAP and CSAW1 consists of a carry save adder, and CSAW2 consists of a 4-2 carry-save adder. Taking the truncation errors into consideration, we should calculate $W[j]$ and $P[j]$ in 2^{-n-c} precision, where $c \approx \log_2 n$.

The cycle time is $t_{DS} + t_{MUL1} + t_{CSAW1} + t_{MUL2} + t_{CSAW2} + t_{load}$, which is about $t_{6CPA} + t_{6comb} + t_{buf} + t_{2MUX} + t_{FA} + t_{2MUX} + t_{2FA} + t_{load}$. Here, t_{6CPA}, t_{6comb}, t_{buf}, t_{2MUX}, t_{FA}, t_{2FA}, and t_{load} are delays for a 6-input carry-propagate adder, a 6-input combinational circuit, a buffer for driving a signal to about $n + c$-bit length, a 2-input multiplexer, a full adder, a 4-2 adder, and register loading, respectively. The cycle time is slightly longer than that of a radix-2 square rooting circuit.

4 A radix-4 version

In this section, we show the details of a radix-4 version of the algorithm and consider its sequential implementation.

4.1 Algorithm

Here, we consider the case that the radix r is 4, the reciprocal square root digit set is $\{-2,-1,0,1,2\}$ (i.e., $a=2$, $\rho=\frac{2}{3}$), and the residual $W[j]$ and the product $P[j]$ is represented in the carry-save form.

The recurrence equations are

$$\begin{aligned} S[j+1] &= S[j]+q_{j+1}4^{-j-1}, \\ W[j+1] &= 4W[j]-(2P[j]+Xq_{j+1}4^{-j-1})q_{j+1}, \\ P[j+1] &= P[j]+Xq_{j+1}4^{-j-1}. \end{aligned}$$

At the beginning of the computation, we let $S[0]$, $W[0]$, and $P[0]$ be as follows. When $X<\frac{3}{8}$, we let $S[0]=2$, $W[0]=1-4X$, and $P[0]=2X$. When $\frac{3}{8}\le X<\frac{3}{4}$, we let $S[0]=\frac{3}{2}$, $W[0]=1-\frac{9}{4}X$, and $P[0]=\frac{3}{2}X$. When $X\ge\frac{3}{4}$, we let $S[0]=1$, $W[0]=1-X$, and $P[0]=X$. Although we may let $S[0]=\frac{3}{2}$, $W[0]=1-\frac{9}{4}X$, and $P[0]=\frac{3}{2}X$ for all X, we use the above values in order to make it possible to use a common digit selection function for all j's.

To obtain a digit selection function, we get the values of $\max_{P\hat{[}j]}(L_k[j])$ and $\min_{P\hat{[}j]}(U_{k-1}[j])$ from (16) and (17).

$$\begin{aligned} \min_{P\hat{[}j]}(U_{-2}[j]) &> -\frac{8}{3}(P\hat{[}j]+2^{-d+1})+\frac{4}{9}X\cdot 4^{-j} \\ \max_{P\hat{[}j]}(L_{-1}[j]) &= -\frac{10}{3}P\hat{[}j]+\frac{25}{36}X\cdot 4^{-j} \\ \min_{P\hat{[}j]}(U_{-1}[j]) &> -\frac{2}{3}(P\hat{[}j]+2^{-d+1})+\frac{1}{36}X\cdot 4^{-j} \\ \max_{P\hat{[}j]}(L_0[j]) &= -\frac{4}{3}P\hat{[}j]+\frac{1}{9}X\cdot 4^{-j} \\ \min_{P\hat{[}j]}(U_0[j]) &= \frac{4}{3}P\hat{[}j]+\frac{1}{9}X\cdot 4^{-j} \\ \max_{P\hat{[}j]}(L_1[j]) &< \frac{2}{3}(P\hat{[}j]+2^{-d+1})+\frac{1}{36}X\cdot 4^{-j} \\ \min_{P\hat{[}j]}(U_1[j]) &= \frac{10}{3}P\hat{[}j]+\frac{25}{36}X\cdot 4^{-j} \\ \max_{P\hat{[}j]}(L_2[j]) &< \frac{8}{3}(P\hat{[}j]+2^{-d+1})+\frac{4}{9}X\cdot 4^{-j} \end{aligned}$$

For $j\ge 1$, from (14), the function has to satisfy

$$\begin{aligned} -\frac{10}{3}P\hat{[}j]+\frac{25}{144}X < m_{-1}(P\hat{[}j]) \le -\frac{8}{3}P\hat{[}j]-\frac{16}{3}\cdot 2^{-d}-2^{-t}, \\ -\frac{4}{3}P\hat{[}j]+\frac{1}{36}X < m_0(P\hat{[}j]) \le -\frac{2}{3}P\hat{[}j]-\frac{4}{3}\cdot 2^{-d}-2^{-t}, \\ \frac{2}{3}P\hat{[}j]+\frac{4}{3}\cdot 2^{-d}+\frac{1}{144}X \le m_1(P\hat{[}j]) \le \frac{4}{3}P\hat{[}j]-2^{-t}, \\ \frac{8}{3}P\hat{[}j]+\frac{16}{3}\cdot 2^{-d}+\frac{1}{9}X \le m_2(P\hat{[}j]) \le \frac{10}{3}P\hat{[}j]-2^{-t}. \end{aligned}$$

Since $P\hat{[}j] > X(X^{-\frac{1}{2}}-\frac{2}{3}4^{-j})-2^{-d+1}$ for $j\ge 1$, we can choose

$$\begin{aligned} m_{-1}(P\hat{[}j]) &= -trunc_4(3(P\hat{[}j]+2^{-6})), \\ m_0(P\hat{[}j]) &= -trunc_4(P\hat{[}j]+2^{-6}), \\ m_1(P\hat{[}j]) &= trunc_4(P\hat{[}j]+2^{-6}), \\ m_2(P\hat{[}j]) &= trunc_4(3(P\hat{[}j]+2^{-6})), \end{aligned}$$

by letting $d=6$ and $t=4$. $trunc_4(\cdot)$ is the truncation of $\cdot$ to 4 fractional bits.

Next, we consider the digit selection for $j=0$. Recall that when $X<\frac{3}{8}$, then $S[0]=2$, $W[0]=1-4X$, $P[0]=2X$, and $4W[0]=4-16X$, that when $\frac{3}{8}\le X<\frac{3}{4}$, then $S[0]=\frac{3}{2}$, $W[0]=1-\frac{9}{4}X$, $P[0]=\frac{3}{2}X$, and $4W[0]=4-9X$, and that when $X\ge\frac{3}{4}$, then $S[0]=1$, $W[0]=1-X$, $P[0]=X$, and $4W[0]=4-4X$. In any case,

$$\begin{gathered} 4W[0] > L_{-1}[0], \quad 4\hat{W}[0] \ge m_{-1}(P\hat{[}0]), \\ \max_{P\hat{[}0]}(L_0[0]) < m_0(P\hat{[}0]) < \min_{P\hat{[}0]}(U_{-1}[0]), \\ \max_{P\hat{[}0]}(L_1[0]) < m_1(P\hat{[}0]) < \min_{P\hat{[}0]}(U_0[0]), \\ 4W[0] < U_1[0], \quad 4\hat{W}[0] \le m_2(P\hat{[}0])-2^{-4} \end{gathered}$$

hold for the above $m_k(P\hat{[}j])$ with $j=0$. Therefore, in any case, we can use the same digit selection function as for $j\ge 1$. q_1 is selected from $\{-1,0,1\}$.

This radix-4 version of the algorithm is summarized as follows. We assume that the on-the-fly conversion is adopted. Since $-2<W[j]<2$, $W[j]$ is represented in a two's complement carry-save form with 2-bit integer part (including the sign bit). Since $0<P[j]<2$, $P[j]$ is represented in an unsigned carry-save form with 1-bit integer part. Therefore, we can select q_{j+1} by examining the most significant 8 (carry-save) bits of $2W[j]$ and the most significant 7 (carry-save) bits of $P[j]$.

Algorithm [RSQRT_R4]

Step 1:

if $X<\frac{3}{8}$ then do
{ $S[0]^{=}=2$; $S[0]^{-}=1$; $W[0]=1-4X$; $P[0]=2X$; }
else if $\frac{3}{8}\le X<\frac{3}{4}$ then do
{ $S[0]^{=}=\frac{3}{2}$; $S[0]^{-}=\frac{1}{2}$; $W[0]=1-\frac{9}{4}X$; $P[0]=\frac{3}{2}X$; }
else do
{ $S[0]^{=}=1$; $S[0]^{-}=0$; $W[0]=1-X$; $P[0]=X$; }

Step 2:

for $j:=0$ to $n-1$ do
{ $q_{j+1}:=$

$$\begin{cases} -2 & if \quad & & 4\hat{W}[j] \le & m_{-1}(P\hat{[}j])-2^{-4} \\ -1 & if & m_{-1}(P\hat{[}j]) \le & 4\hat{W}[j] \le & m_0(P\hat{[}j])-2^{-4} \\ 0 & if & m_0(P\hat{[}j]) \le & 4\hat{W}[j] \le & m_1(P\hat{[}j])-2^{-4} \\ 1 & if & m_1(P\hat{[}j]) \le & 4\hat{W}[j] \le & m_2(P\hat{[}j])-2^{-4} \\ 2 & if & m_2(P\hat{[}j]) \le & 4\hat{W}[j] \le & \end{cases};$$

($4\hat{W}[j]$: truncation of $4W[j]$ to 4 fractional bits.)
($\hat{P[j]}$: truncation of $P[j]$ to 6 fractional bits.)

$$\begin{pmatrix} m_{-1}(\hat{P[j]}) = -trunc_4(3(\hat{P[j]} + 2^{-6})) \\ m_0(\hat{P[j]}) = -trunc_4(\hat{P[j]} + 2^{-6}) \\ m_1(\hat{P[j]}) = trunc_4(\hat{P[j]} + 2^{-6}) \\ m_2(\hat{P[j]}) = trunc_4(3(\hat{P[j]} + 2^{-6})) \end{pmatrix}$$

Calculate $S[j+1]^=$ and $S[j+1]^-$;
(On-the-fly conversion.)
$W[j+1] := 4W[j] - (2P[j] + Xq_{j+1}4^{-j-1})q_{j+1}$;
(Carry-save additions.)
$P[j+1] := P[j] + Xq_{j+1}4^{-j-1}$;
(Carry-save addition.)
}

$S[n]^=$ is the result.

4.2 Sequential implementation

Here, we consider implementation of the radix-4 version as a sequential circuit which performs one iteration of Step 2 in each clock cycle. The circuit structure is the same as that of the sequential implementation of the radix-2 version shown in Fig. 2, except that DS is fed with most significant bits of $P[j]$ as well.

DS is a combination of an 8-bit carry-propagate adder, a 7-bit carry-propagate adder and a 15-input combinational circuit. A buffer for driving q_{j+1} is also required. OTFC mainly consists of a pair of 2-to-1 selectors. MUL1 and MUL2 are multiplexers, which produce 0 or the data input itself or its complement or the double of the data input or its complement according to the value of q_{j+1}. Each of CSAP and CSAW1 consists of a carry save adder, and CSAW2 consists of a 4-2 carry-save adder.

The cycle time is again $t_{DS} + t_{MUL1} + t_{CSAW1} + t_{MUL2} + t_{CSAW2} + t_{load}$, which is about $t_{8CPA} + t_{15comb} + t_{buf} + t_{4MUX} + t_{FA} + t_{4MUX} + t_{2FA} + t_{load}$. Here, t_{8CPA}, t_{15comb}, and t_{4MUX} are delays for an 8-input carry-propagate adder, a 15-input combinational circuit, and a 4-input multiplexer, respectively. The cycle time is slightly longer than that of a radix-4 square rooting circuit.

5 Conclusion

We have proposed a digit-recurrence algorithm for computing the reciprocal square root which appears frequently in multimedia and graphics applications. In the proposed algorithm, in order to remove a large multiplication from the residual calculation, we keep the product of the partial result and the operand. The algorithm computes the reciprocal square root by iteration of carry-propagation-free additions, shifts, and multiplications by one digit. We can design different specific versions of the algorithm, depending on the radix, the redundancy factor of the digit set, and etc. We have designed a radix-2 and a radix-4 version and consider their sequential implementations. The implementations have a regular structure with bit-slice feature, and are suitable for VLSI.

Providing a reciprocal square rooting circuit based on the proposed algorithm, we can perform a reciprocal square root operation in about the same latency as division or square rooting, without using a multiplier. Combining a reciprocal square rooting circuit with a divider and a square rooting circuit based on digit recurrence algorithms to form a divide/square-root/reciprocal-square-root unit should be a practical implementation.

References

[1] C. Price: 'MIPS IV Instruction Set, revision 3.2,' MIPS Technologies Inc., Sep. 1995.

[2] Silicon Graphics Inc.: 'MIPS RISC Technology R10000 Microprocessor Technical Brief,' www.sgi.com/processors/r10k/tech_info/Tech_Brief.html.

[3] International Business Machines Co.: 'PowerPC 740, PowerPC 750 RISC Processor User's Manual,' Feb. 1999.

[4] 'Fast hardware-based algorithms for elementary function computations using rectangular multipliers,' IEEE Trans. Comput., vol. C-43, no. 3, pp. 278–294, Mar. 1994.

[5] E. Antelo, T. Lang and J. D. Bruguera: 'Computation of $\sqrt{x/d}$ in a very high radix combined division/square-root unit with scaling and selection by rounding,' IEEE Trans. Comput., vol. C-47, no. 2, pp. 152–161, Feb. 1998.

[6] N. Takagi: 'Powering by a table look-up and a multiplication with operand modification,' IEEE Trans. Comput., vol. C-47, no. 11, pp. 1216–1222, Nov. 1998.

[7] M. D. Ercegovac, T. Lang, J.-M. Muller and A. Tisserand: 'Reciprocation, square root, inverse square root, and some elementary functions using small multipliers,' IEEE Trans. Comput., vol. C-49, no. 7, pp. 628–637, July 2000.

[8] M. D. Ercegovac and T. Lang: *Division and Square Root – Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994.

[9] M. D. Ercegovac and T. Lang: 'On-the-fly conversion of redundant into conventional representations,' IEEE Trans. Comput., vol. C-36, no. 7, pp. 895–897, July 1987.